COMP4161: Advanced Topics in Software Verification

INV

Gerwin Klein, June Andronick, Ramana Kumar, Miki Tanaka

S2/2017

data61.csiro.au

Last Time

- ➜ Weakest preconditions
- ➜ Verification conditions
- ➜ Example program proofs
- ➜ Arrays, pointers

Content

- ➜ Intro & motivation, getting started [1]
- ➜ Foundations & Principles
  - Lambda Calculus, natural deduction [1,2]
  - Higher Order Logic [3[a]]
  - Term rewriting [4]
- ➜ Proof & Specification Techniques
  - Inductively defined sets, rule induction [5]
  - Datatypes, recursion, induction [6, 7]
  - Hoare logic, proofs about programs, C verification [8[b],9]
  - (mid-semester break)
  - Writing Automated Proof Methods [10]
  - Isar, codegen, typeclasses, locales [11[c],12]

---

[a] a1 due; [b] a2 due; [c] a3 due

Today

Practice with invariants!

Recall:

- ➜ it needs to be an invariant
- ➜ it needs to be enough

Example 1

$\{\ a \geq 0\ \wedge\ b \geq 0\ \}$
$A := 0;$
$B := 0;$ $\qquad 0 = b * 0$
INV $\{$ B = b * A$\}$
WHILE $A \neq a$ $\qquad B = b * A \wedge A \neq a \longrightarrow B + b = b * (A + 1)$
DO
$\quad B := B + b;$
$\quad A := A + 1$
OD $\qquad B = b * A \wedge A = a \longrightarrow B = b * A$
$\{\ B = b * a\ \}$

Example 2

$\{\ a \geq 0\ \wedge\ b \geq 0\ \}$
$A := a;$
$B := 1;$ $\qquad 1 = b^{a-a}$
INV $\{\ B = b^{a-A}\}$
WHILE $A \neq 0$ $\qquad B = b^{a-A} \wedge A \neq 0 \longrightarrow\ B * b = b^{a-(A-1)}$
DO
$\quad B := B * b;$
$\quad A := A - 1$
OD $\qquad B = b^{a-A} \wedge A = 0\ \longrightarrow\ B = b^{a}$
$\{\ B = b^{a}\ \}$

Example 4

Try with $b = 12 = 2^2 + 2^3$ or $b = 2^1 + 2^3$...

$\{\ a \geq 0 \wedge b \geq 0\ \}$

$A := a;\ B := b;\ C := 1;$ $\quad a^b = 1 * a^b$

INV $\{\ a^b = C * A^B\}$

WHILE $B \neq 0$ $\quad a^b = C * A^B \wedge B \neq 0 \longrightarrow\ a^b = (C * A) *$

DO

INV $\{\ a^b = C * A^B\}$

WHILE (B mod 2 = 0)

$a^b = C * A^B \wedge B\ mod\ 2 = 0 \longrightarrow\ a^b = C * (A * A)^{B\ di}$

DO

$A := A * A;$

$B := B\ div\ 2;$

OD

$C := C * A;$

$B := B - 1$

Example 5

$\{\ True\ \}$
$X := x;$
$Y := [];$ $\qquad (rev\ x)@[] = rev\ x$
INV $\{\ (rev\ X)@Y = rev\ x\}$
WHILE $X \neq []$
$\qquad (rev\ X)@Y = rev\ x\ \wedge X \neq [] \longrightarrow$
$\qquad (rev\ (tl\ X))@((hd\ X)\#Y) = rev\ x$
DO
$\quad Y := (hd\ X\#Y);$
$\quad X := tl\ X$
OD $\qquad (rev\ X)@Y = rev\ x\ \wedge X = [] \longrightarrow Y = rev\ x$
$\{\ Y = rev\ x\ \}$

Example 6

$LEQ\ A\ n = \forall k.\ k < n \longrightarrow\ A!k \leq piv$
$QEQ\ A\ n = \forall k.\ n < k < length\ A \longrightarrow A!k \geq piv$
$EQ\ A\ n\ m = \forall k.\ n < k < m \longrightarrow A!k = piv$

$\{\ 0 < length\ A\ \}$
$l := 0; u := length\ A - 1;$
INV $\{\ LEQ\ A\ l \wedge GEP\ A\ u \wedge u < lengthA \wedge l \leq lengthA\}$
WHILE $l \leq u$
DO
  INV $\{\ LEQ\ A\ l \wedge GEP\ A\ u \wedge u < lengthA \wedge l \leq lengthA\}$
  WHILE $l < length\ A \wedge A!l \leq piv$ DO $l := l + 1$ OD;

  INV $\{\ LEQ\ A\ l \wedge GEP\ A\ u \wedge u < lengthA \wedge l \leq lengthA\}$
  WHILE $0 < u \wedge piv \leq A!u$ DO $u := u - 1$ OD;

  IF $l \leq u$ THEN $A := A[l := A!u, u := A!l]$ ELSE SKIP FI
OD
$\{\ LEQ\ A\ u \wedge EQ\ u\ l \wedge GEP\ A\ l\ \}$

Example 7

Reminder:

datatype ref = Ref int | Null

Pointer access: p→field

Pointer update: p→field :== v

Definition:

"*List nxt p Ps*" is a linked list, starting at pointer *p* following the next pointer through the function *nxt*, and where *Ps* contains the list of the pointers of the linked list.

$\{\ List\ nxt\ p\ Ps \wedge X \in Ps\ \}$ $\quad \exists Qs.\ List\ nxt\ p\ Qs \wedge X \in Qs$

INV $\{\ \exists Qs.\ List\ nxt\ p\ Qs \wedge X \in Qs\}$

WHILE $p \neq Null \wedge p \neq Ref\ X$ $\quad \exists Qs.\ List\ nxt\ p\ Qs \wedge X \in Qs$

$\wedge p \neq Null \wedge p \neq Ref\ X \longrightarrow$

$\exists Qs.\ List\ nxt\ (p \rightarrow nxt)\ Qs \wedge X \in Q$

DO

$p := p \rightarrow nxt;$

Example 8

What is is Isabelle function doing?

```
fun f :: 'a list ⇒' a list ⇒' a list where
  f [] ys = ys|
  f xs [] = xs|
  f (x#xs) (y#ys) = x#y# f xs ys
```


Example 8

What is is Isabelle function doing?

```
fun splice :: 'a list ⇒' a list ⇒' a list where
  splice [] ys = ys|
  splice xs [] = xs|
  splice (x#xs) (y#ys) = x#y# f xs ys
```

Let's write it with linked lists!

Example 8

List nxt p Ps = *Path nxt p Ps Null*

Path nxt p Ps Null is a linked list from p to q following function *nxt* and containing list of pointers *Ps*

$\{\ List\ nxt\ p\ Ps \wedge List\ nxt\ q\ Qs \wedge (set\ Ps \cap set\ Qs) = \{\} \wedge size\ Qs \leq size\ Ps$

$pp := p;$

INV $\{\ \exists PPs\ QQs\ PPPs.\quad size\ QQs \leq size\ PPs \wedge$

$List\ nxt\ pp\ PPs \wedge List\ nxt\ qq\ QQs \wedge Path\ nxt\ p\ PPPs\ pp$

$\wedge\ PPPs@splice\ PPs\ QQs = splice\ Ps\ Qs \wedge$

$set\ PPs \cap set\ QQs = \{\} \wedge distinct\ PPPs \wedge set\ PPPs \cap (set\ PPs \cup set\ QQs$

$\}$

WHILE $q \neq Null$

DO

$qq := q \rightarrow nxt; q \rightarrow nxt := pp \rightarrow nxt; pp \rightarrow nxt = q; pp := q \rightarrow nxt; q :=$

OD

$\{\ List\ nxt\ p\ (splice\ Ps\ Qs)\ \}$

Demo